5-30-2018

Poster: Towards safe refactoring for intelligent parallelization of Java 8 streams

Yiming Tang
CUNY Graduate Center

Raffi T. Khatchadourian
CUNY Hunter College

Mehdi Bagherzadeh
Oakland University

Syed Ahmed
Oakland University

How does access to this work benefit you? Let us know!
Follow this and additional works at: https://academicworks.cuny.edu/hc_pubs
Part of the Programming Languages and Compilers Commons, and the Software Engineering Commons

Recommended Citation

This Poster is brought to you for free and open access by the Hunter College at CUNY Academic Works. It has been accepted for inclusion in Publications and Research by an authorized administrator of CUNY Academic Works. For more information, please contact AcademicWorks@cuny.edu.
Towards Safe Refactoring for Intelligent Parallelization of Java 8 Streams

Yiming Tang\textsuperscript{1} Raffi Khatchadourian\textsuperscript{1} Mehdi Bagherzadeh\textsuperscript{2} Syed Ahmed\textsuperscript{2}
\textsuperscript{1}City University of New York (ponder@hunter.cuny.edu) \textsuperscript{2}Oakland University

\section*{Introduction}

The Java 8 Stream API sets forth a promising new programming model that incorporates functional-like, MapReduce-style features into a mainstream programming language.

\section*{Problem}

- Developers must \textit{manually} determine whether running streams in parallel is efficient yet interference-free.
- Using streams correctly and efficiently requires many subtle considerations that may not be immediately evident.
- Manual analysis and refactoring can be error- and omission-prone.

\section*{Automated Tool}

Our Eclipse Plug-in, based on ordering and typestate analysis, automatically identifies and executes refactoring opportunities where improvements can be made to Java 8 Stream code. The parallelization is “intelligent” as it carefully considers each context and may actually result in de-parallelization.

\section*{Contributions}

We devise an automated refactoring approach that assists developers in writing optimal stream code. The approach determines when it is safe and advantageous to convert streams to parallel and optimize parallel streams. A case study is performed on the applicability of the approach.

\section*{Refactorings}

- **Convert Sequential Stream to Parallel.** Determines if it is advantageous and safe to convert a sequential stream to parallel.
- **Optimize Parallel Stream.** Decides which transformations can improve the performance of a parallel stream, including unordering and converting to sequential.

\section*{Typestate Analysis}

Our in-progress approach uses typestate analysis to determine stream attributes when a terminal operation is issued. A typestate variant is being developed since operations like \texttt{sorted()} return (possibly) new streams derived from the receiver with their attributes altered. Labeled transition systems (LTSs) are used for execution mode and ordering.

\section*{Preliminary Experimental Results}

<table>
<thead>
<tr>
<th>Projects</th>
<th>Candidate Streams</th>
<th>Refactorable Streams</th>
</tr>
</thead>
<tbody>
<tr>
<td>threethen-extra</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>jOOQ</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>dani</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>japhyx</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>bootique</td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>jdk8-experiments</td>
<td>16</td>
<td>4</td>
</tr>
<tr>
<td>htm.java</td>
<td>21</td>
<td>7</td>
</tr>
<tr>
<td>jetty-project</td>
<td>22</td>
<td>7</td>
</tr>
<tr>
<td>streamsql</td>
<td>22</td>
<td>2</td>
</tr>
<tr>
<td>java-design-patterns</td>
<td>28</td>
<td>17</td>
</tr>
<tr>
<td>Grand Total</td>
<td>128</td>
<td>42</td>
</tr>
</tbody>
</table>

Table: Preliminary results summary.

\section*{Conclusion}

We have developed an automated refactoring approach that “intelligently” optimizes Java 8 stream code. Based on ordering and typestate analysis, it automatically deems when it is safe and advantageous to run stream code either sequentially or in parallel.